TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

RISTO AUTIO

AUTOMATED TESTING OF CROSS-PLATFORM MOBILE APPLICATIONS

Master of Science thesis

Examiner: Prof. Tommi Mikkonen
Examiner and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on 3rd February 2016

ABSTRACT

RISTO AUTIO: Automated testing of cross-platform mobile applications
Tampere University of Technology
Master of Science thesis, 46 pages
April 2016
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiner: Prof. Tommi Mikkonen
Keywords: Cross-platform, PhoneGap, Testing, Android, iOS

Mobile applications are becoming more common as the number of mobile devices grows. For these devices there are a number of operating systems that run applications that have been made for them. Implementing an application for multiple platforms has commonly required creating multiple implementations in order to run the application on each of the desired platforms. This has lead to the development of cross-platform mobile applications, which allow writing one implementation that can be used for multiple platforms.

In this thesis, the intent is to evaluate if there are tools for automating testing cross-platform mobile applications, that are viable for using for testing mobile applications developed by Dicode Ltd. The tool used for developing cross-platform mobile applications is PhoneGap.

This thesis evaluates three available tools for testing cross-platform mobile applications. The target platforms in this evaluation are Android and iOS. A set of criteria are used to evaluate the frameworks.

The results of this thesis recommend the use of a framework called Calabash for automating the testing of cross-platform mobile applications. Calabash performed well with all of the evaluation criteria and it is able to test Android and iOS applications. These are the two most popular operating systems for smartphones.

TIIVISTELMÄ

RISTO AUTIO: Alustariippumattomien mobiilisovellusten testauksen automatisoiminen
Tampereen teknillinen yliopisto
Diplomityö, 46 sivua
Huhtikuu 2016
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastajat: Prof. Tommi Mikkonen
Avainsanat: Alustariippumattomuus, Android, iOS, PhoneGap, Testaus

Mobiilisovellukset ovat yleistyneet samalla kun älypuhelinten määrä on kasvanut. Näille puhelimille on useita käyttöjärjestelmiä. Tämän seurauksena mobiilisovelluskehittäjät, jotka ovat tuottaneet sovelluksen usealle käyttöjärjestelmälle, ovat joutuneet toteuttamaan saman sovelluksen useita kertoja. Alustariippumattomat sovellukset pyrkivät ratkaisemaan tämän ongelman mahdollistamalla saman toteutuksen käyttämisen useammalla kohdealustalla.

Tässä työssä pyritään löytämään työkalu Dicode Oy:n kehittämien alustariippumattomien mobiilisovellusten testauksen automatisoimiseksi. Mobiilisovellusten kehittämiseen käytetty työkalu on PhoneGap.

Tämä työ arvioi kolme eri mobiilisovellusten testauksen automatisoivaa työkalua. Käyttöjärjestelmät joilla arviointi tehdään ovat Android ja iOS. Työssä esitetään joukko arviointiin käytettyjä kriteerejä ja arvioinnin tulokset.

Työn tuloksien perusteella esitetään alustariippumattomien mobiilisovellusten testauksen automatisointiin Calabash-testauskehystä. Calabash sai hyvät tulokset käytetyille kriteereille ja se kykenee testaamaan sekä Android- että iOS-sovelluksia.

PREFACE

I want to thank my friends and family for their support over the years. Also my colleagues at Dicode for their support. Thank you to my supervisor, Janne Sikiö for his comments, support, and for providing me time for writing this thesis. I also want to thank my examiner, professor Tommi Mikkonen for his valuable feedback during this period.

Tampere 20.3.2016

Risto Autio

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND SYMBOLS

ADB	Android Debug Bridge
API	Application programming interface
BDD	Behavior driven development
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DSL	Domain Specific Language
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated development environment
REST	Representational State Transfer
SDK	Software development kit
STaaS	Software testing as a service
UI	User Interface
XPath	XML Path Language

1. INTRODUCTION

Smartphones have grown in popularity over the recent years. This is due to the applications they offer, and the number of applications has increased with the number of smartphones in the market. Applications are used to meet different needs and they are being used commonly to perform different tasks. As it is commonly expected for companies to have web pages, it is becoming more common to expect them to have mobile applications. However, due to the number of different mobile platforms, it is not as easy to offer a mobile application on multiple platforms.

One solution is to create one implementation that works on multiple platforms. This is known as cross-platform development and it prevents the need to create an implementation for each of the targeted platforms. While cross-platform mobile applications have grown in popularity, testing these applications is not possible with the tools used for testing native applications. However, extensive testing of mobile applications requires testing an application on multiple devices that have different screen sizes, and which behave in different ways.

Due to the importance of testing applications, there are some tools that have been created for testing cross-platform mobile applications. However, these tools are new and they are still being developed. Mobile operating systems are updated frequently and these tools need to be updated as well, because new features are added to the operating systems or old features are updated.

In this thesis, the intent is to evaluate if there are tools for automating testing cross-platform mobile applications, that are viable for using for testing mobile applications developed by Dicode Ltd. Dicode Ltd is a company that has mainly focused on web application development, and cross-platform applications have been seen as a solution for developing mobile applications. Cross-platform mobile applications have the benefit of using wide spread web technologies, which makes it possible for web application developers to develop and maintain these mobile applications.

This thesis describes the solution offered by cross-platform application development, and evaluates if there is a suitable tool for testing these applications. In chapter 2 the concepts of developing cross-platform mobile applications are introduced. In chapter 3 the methods for testing mobile applications are listed. In chapter 4 the frameworks to be evaluated are introduced. The criterion for evaluating the frameworks are introduced in chapter 5. And in chapter 6 the results of the evalution are presented. Finally in chapter 7 the conclusions of the thesis are presented.

2. CROSS-PLATFORM MOBILE APPLICATION DEVELOPMENT

Developing mobile applications for a large audience requires deploying them on multiple platforms. So far there have been two approaches for developing the same application for multiple platforms. The first is to develop the application for one platform at a time. This approach requires developers to learn the use of the tools needed for the target platform. This way developing for multiple platforms ends up taking a lot of time. Another solution has been to divide developers into teams that develop the application for different platforms at the same time. This approach is faster than the first one, but the number of developers needed will be greater and so will the cost of development. [23]

Cross-platform development solutions attempt to provide a better alternative for developing applications for multiple platforms. First in section 2.1 we will first introduce mobile operating systems and examine the two most common operating systems used by mobile devices. In section 2.2 we will look at different approaches to cross-platform development for mobile applications. In section 2.3 we will focus on the differences between the applications created using these approaches and native mobile applications. Finally, we will introduce PhoneGap, the cross-platform development tool used to examine cross-platform development in this thesis in section 2.4.

2.1 Mobile operating systems

There are many operating systems for mobile devices, but the Android operating system developed by Google is the most popular one. It has a worldwide market share of over 80%. The second most popular operating system for mobile devices is iOS which is developed by Apple and its worldwide market share is over 15%. The sale of smartphones grew rapidly after they became available, but the rate has been

slowing down and predictions say that Android and iOS will maintain about the same market shares in 2019 as they do now. [35]

Because of their popularity, Android and iOS are the default target operating systems when developing mobile applications. Both operating systems will be introduced in the next subsections.

2.1.1 Android

Android is a mobile operating system developed by Google. It is based on the Linux kernel and it is open source. Therefore it is possible for anyone to customize the operating system for their own needs. This makes it a popular operating system with many technology companies. [7]

The development of the Android operating system is supported by the Open Handset Alliance. The Open Handset Alliance consist of different technology companies, such as Google, Intel, Motorola, and Sprint. The Open Handset Alliance is not publicly open, but companies join by a closed process managed by Google. Many of the Open Handset alliances members have contributed intellectual property to the Android project. [29]

Android applications can be developed on platforms that support the Android SDK. These are Windows, Mac OS and Linux. Different integrated development environments (IDE) can be used for developing applications for Android. Applications for Android are developed using Java which is compiled into bytecode and translated into the Android platforms own byte code. [8]

The distribution of different Android versions can be seen in Table 2.1. Devices using an older version than Android 2.2 are not included. Google estimates that Android versions older than 2.2 account for about 1% of devices. The data has been collected by Google using their Play Store application [27], so it contains only the data from devices that have the application installed. [5]

Android's major release versions are identified using codenames. Each Android platform version supports one API level and each application has a minimum required API level. The higher API levels are designed so that they are compatible with all earlier versions, and while old parts of the API get deprecated they are not removed. This makes it possible for existing applications to still use the old parts. [10]

Table 2.1 *Distribution of Android versions [5]*

Version	Codename	API	Distribution
2.2	Froyo	8	0.2%
2.3.3 - 2.3.7	Gingerbread	10	3.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2.9%
4.1.x	Jelly Bean	16	10.0%
4.2.x	Jelly Bean	17	13.0%
4.3	Jelly Bean	18	3.9%
4.4	KitKat	19	36.6%
5.0	Lollipop	21	16.3%
5.1	Lollipop	22	13.2%
6.0	Marshmallow	23	0.5%

In addition to the many different Android versions, Google lists 38 different vendors that produce devices running the Android operating system [9]. Distributing Android applications can be done by using marketplaces. The most used marketplace for Android applications is the Google Play Store [26], but other marketplaces such as Amazon's app store [4] can be used for distributing Android applications.

The Activity lifecycle describes how an activity in Android can switch between different states. In Android applications an activity component is used to manage what the user sees and it is also used to handle the input given by the user. Applications usually contain a number of activities, one for each view. The lifecycle of an activity can be seen in Figure 2.1. All applications have a main activity which is created when the application is started. When an activity is started, it executes three methods before it is able to interact with the user. Activities create other activities, and the created activity becomes the running activity, while the old activity moves to the foreground and is no longer active. The stopped activity maintains its state, so it is possible to continue that activity if the user returns to it. [11]

2.1.2 iOS

The iOS operating system is developed by Apple, and it is used in iPhone and iPad devices. While Android makes it possible to use many tools and different IDEs for development, Apple limits the use of other tools. Xcode is used as the IDE when developing iOS applications, and it handles compiling, validating, and sending the application to the Apple App Store. The same IDE can be used for debugging, as

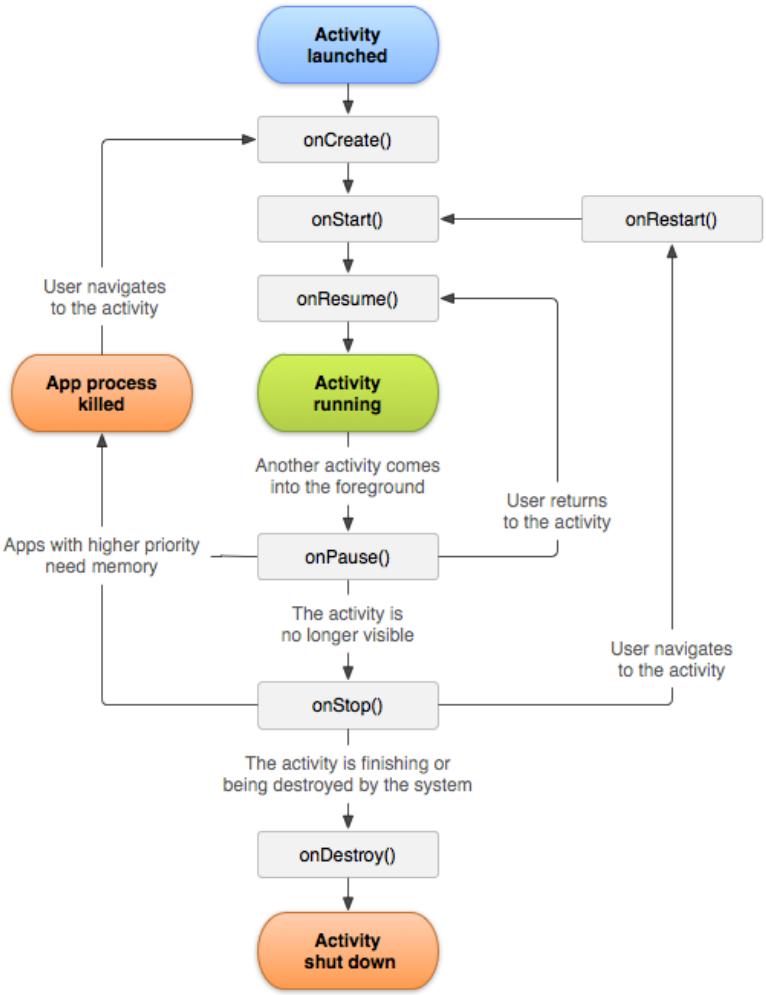

Figure 2.1 *The lifecycle of an Android activity [11].*

well as analyzing the applications memory usage and its performance. [17]

In order to install an application on an iOS device during development, the applications needs to be signed using signing credentials from Apple. The signing credentials need to be purchased from Apple by the developer. Without the credentials developers are still able to test their applications on the emulators included in Xcode. [17]

Unlike in Android, iOS applications have more demanding design guidelines. It is possible for applications to be rejected from the Apple App Store if the application does not follow the development guidelines. It is important for applications to look and feel like native applications even if they are hybrid applications that do not use

native components for the user interface. [15]

The distribution of different iOS versions limited to fewer version than with Android. The majority of Apple devices using iOS are using iOS 9 with a share of 70%. 22% of devices are using iOS 8, and the rest are using earlier versions of iOS. Distributing applications for iOS is done using Apple App Store, and in order to get the application in to distribution it has to go trough a review process. Submitting or updating an application to the App Store can take more than a week. This means that even if a critical bug is found in an application running on iOS, getting the fixed application to the end user can take a lot of time. [16]

The lifecycle of an iOS application consist of five states, which are not running, inactive, active, background, and suspended. These states and the paths the application can take can be seen in Figure 2.2. The application is in the not running state if the system has terminated the application in order to reclaim the resources used by the application or because the user has not started application. Applications are usually inactive only when they are transitioning to other states. If an application is inactive it cannot receive events, but it is on the foreground. While the application is in the active state it is running normally and receiving events. Applications that are in the background state can still execute code. Applications can also be launched directly into the background. Most of the applications are in the background state only when they are ready to be suspended. Suspended applications are in the background and they do not execute any code. They maintain the state they had in the background state, but it is possible for the system to purge the application if the system is in need of memory. [19]

Applications are terminated either by the user or the operating system. The application is usually in the suspended state when terminated, but the operating system can terminate an application that is not responding as expected and it may be in some other state. The suspended application is not notified when it is terminated, and this is why it should not be expected to do any operation before terminating. [19]

2.2 Towards cross-platform application development

Cross-platform application development attempts to solve the problem of having to write a different implementation for each target platform. There are a number

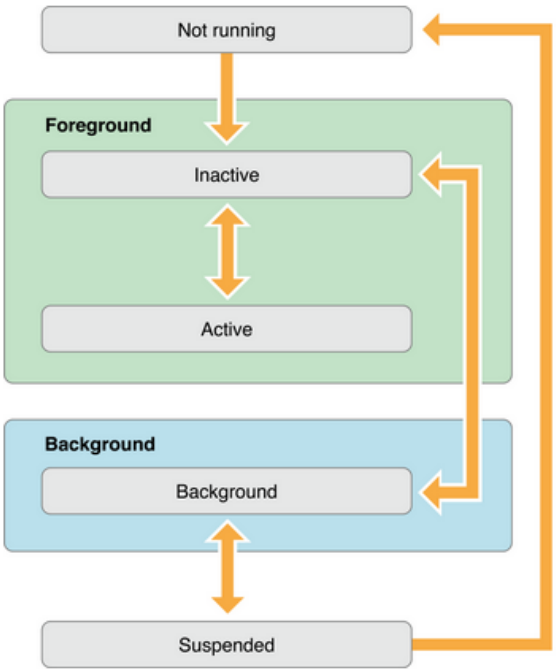

Figure 2.2 *State changes in an iOS application [19].*

of ways to use the same code base to build applications running on different platforms. The approaches towards developing cross-platform mobile applications can be divided into four categories. These are web, generated, interpreted, and hybrid applications. In the following, these approaches and how they are used will be described. [44]

With the web approach, a mobile devices' web browser is used to open a web application that has been made with standard web technologies. While this approach is platform independent, it does not offer any of the devices' native functions and the applications performance is slower than with the other approach methods. The resulting application cannot be used offline and it is not distributed through any application store. [34]

Generated applications are created by generating a code base for each target platform. The generated code is then compiled to build a native application. Since this results in a fully native application, there are no issues with the look and feel of the application. Also the performance of the application is not compromised since the result is a native application. The generated code is not optimized, but it is possible to edit the code base. However making changes to the generated code can be

difficult due to the structure of the application. Complex changes can also require a good understanding of all the platforms where the changes would be applied to. [44]

Interpreted applications use a virtual machine which allows the same code to be interpreted on different platforms. Applications running on virtual machines are slower than applications running native code, but they are easier to maintain. This approach does not make it easy to extend the applications' functionality, if a feature has not been implemented by the virtual machine. [32]

Hybrid applications are built using mainly popular web technologies such as HyperText Markup Language (HTML), JavaScript and Cascading Style Sheets (CSS). The use of popular technologies make it easier for developers to adapt as it does not have a learning curve. The resulting applications embed a HTML application in the platforms native WebView component. This is usually the only native component used by hybrid applications and therefore achieving a native look and feel is the responsibility of the developer. The aim of hybrid applications is to combine the advantages of web and native applications. In this thesis hybrid applications will be used for examining cross-platform applications. [44]

IBM uses a spectrum to divide hybrid application development approaches into four types. The approach types and their position in the web-native continuum can be seen in Figure 2.3. The first approach is a native shell that encloses an existing mobile website. This approach is the closest to a traditional mobile website but it still has the advantage of giving the website access to the mobile devices native functionality. [34]

The second and most common approach in hybrid application development is to prepackage web resources. In this approach HTML, JavaScript and CSS files are packaged into the application so they are not loaded from an external source. This makes it possible for the application to work offline the same way as a native application. This also improves performance and the application can appear more native due to the improved responsiveness. [34]

The two last approaches use a mixture of native and HTML screens. For example the application may start in a native screen and use a WebView for part of the applications functionality. This makes it possible to use native capabilities and improve performance when needed. This approach is more difficult for developers to

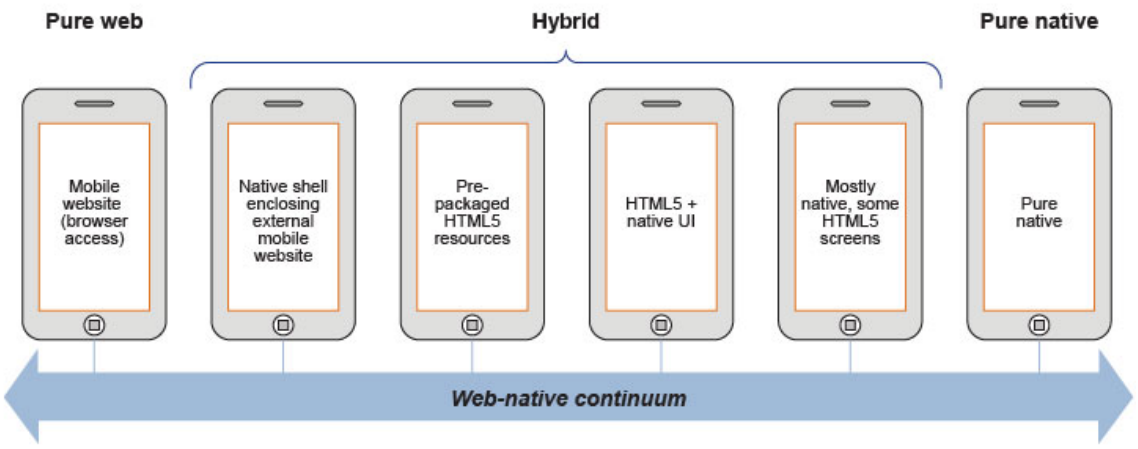

Figure 2.3 *Spectrum of mobile app development approaches [34].*

adapt as it requires a deeper understanding of the platform and the resulting code base is not as portable compared to the previous approaches. [34]

2.3 Comparison to native applications

Xanthopoulos and Xinogalos consider cross-platform mobile development to be the best alternative solution for companies that need to target multiple platforms since the concept of writing code once and running it anywhere cannot be applied to native applications. They also note that cross-platform development will save time and effort in development and also simplify maintenance and deployment. [44]

2.3.1 Overview

Xanthopoulos and Xinogalos use a set of characteristics for comparing different tools for cross-platform development. These characteristics also describe some of the main benefits gained by cross-platform development tools and some of the challenges they try to overcome. The first characteristic evaluates if it is possible to distribute an application in marketplaces and how easy it is. Applications need to be compiled for the target platform in order to be distributed in the marketplaces. The second characteristic evaluates if it possible to use widespread technologies to develop the application. Commonly used technologies make it easier for developers and companies to start using the available tool. [44]

The third characteristic evaluates the applications ability to access the devices hardware and data. The fourth characteristic evaluates if the application uses a native user interface or if a user interface that has a native look and feel is possible to simulate. The last characteristic evaluates the performance perceived by the end user. Many mobile applications perform actions that do not require much processing power, and the performance can look as if the application was a native application. [44]

2.3.2 Challenges

Offering native performance while offering a native look and feel at the same time is one of the most challenging aspects of developing cross-platform mobile applications. With some approaches it is also difficult to reach certain application marketplaces, as applications that do not comply with development guidelines can be rejected from them. Implementing a user interface for a cross-platform application that follows development guidelines for all of the targeted platforms can be a challenge. [44]

Since cross-platform applications run on different operating systems, many of the challenges are caused by the differences between these operating systems e.g. accessing the file system, communication on-line, etc. Also the operating systems have features that often need to be treated differently such as touch interaction, hardware management, screen orientation, soft keyboard data entry, etc. [20]

Compiling a mobile application is done using an operating system that has support for the target platform. This means that in order to compile an application for iOS it needs to be done using a MAC. For some platforms such as Android the compilation can be done on many different platforms. [21]

Updates to the mobile devices operating system can alter the behaviour of a cross-platform application. Hence updated operating systems may require changes to the application. Therefore it is possible that cross-platform applications may require more maintenance and more frequent updates. [23]

Creating automated tests for cross-platform applications is also a challenge no matter which approach is used for creating the application. Since the generated applications and applications running on virtual machines use native components in the interface, they require their own tests for each platform using the platforms tools.

With hybrid applications the same tools are not available since the user interface is built using the WebView and not using any other native components.

2.3.3 Benefits

Amatya and Kurti [3] consider fragmentation to be the most prominent challenge when developing mobile applications. Fragmentation can either refer to hardware-based fragmentation or software-based fragmentation. In hardware-based fragmentation an application can run in the same operating system but on many devices that have different screen sizes, graphics cards and processors. In software-based fragmentation there are differences in either the operating system or in the software it is running. In the case of Android many vendors customize the Android version they use for each device they provide. Some mobile phone carriers also offer software customization. Cross-platform development aims to solve some of the problems caused by fragmentation. [31]

Developers need to have a good understanding of the target platform when developing a native application. For each platform there is a software development kit (SDK) that provides the required tools for developing applications. Using the SDKs requires the use of a certain programming language in each case. Android applications are developed using Java and the development of iOS applications require the use of Objective-C. [44]

Development frameworks for cross-platform applications can have the benefit of offering more widespread technologies for the development process. This makes it more likely that a developer does not need to learn new technologies that may not be used elsewhere. The use of widespread technologies can then save time and effort during development. [21]

Since implementing cross-platform applications is faster than native ones, they are also useful for quick prototypes of applications that may be implemented as native applications for multiple platforms [3]. Already existing web applications are also possible to reuse as the basis for a mobile application and it may only require adding extra functionality to the already implemented web application.

Targeting to more than one platform also has the benefit of the developed application being available on multiple market places. This way the application is available to a larger target audience. [20]

2.4 PhoneGap

PhoneGap is one of the most popular development frameworks for building hybrid mobile applications. It supports the most common operating systems such as Android, iOS, Windows Mobile, Blackberry and many more. [44] When comparing different cross-platform development tools Appiah et al. [13] rated PhoneGap as the best tool especially when comparing its development speed and capability. PhoneGap applications are built using standard web technologies such as HTML, JavaScript and CSS [12]. The code is then reusable on different platforms, and it has access to the native devices Application Programming Interface (API). There is less of a learning curve for PhoneGap than there is for writing code for Android or iOS. PhoneGaps' architecture can be seen in Figure 2.4. [22]

The web application inside the PhoneGap application is the only part of the application the developer has to implement. In addition to the common web technologies it contains resources such as images, fonts, and audio files. The mobile operating system is responsible for managing the users' input and the devices sensors. Graphics are used to display information to the user and services refer to the operating systems services used to access the underlying hardware. The WebView and PhoneGaps plugins are described in more detail in the following subsections. [22]

Applications created with PhoneGap are called hybrid applications, because they combine features of web applications and of native applications [23]. The only native component when building an application with PhoneGap is the WebView that is embedded into the native app. This allows the application code written in JavaScript to be used on any device. [32]

2.4.1 WebView

User interaction with an application built with PhoneGap interacts with the user using an embedded browser, which is known as a WebView. Because of this there are no native components provided by the framework and therefore PhoneGap does not offer a native look and feel for the application. Therefore PhoneGap is commonly used with User interface (UI) libraries to present the application to the user with a more native look compared to traditional web applications. [32]

The browser component is used to interact with the user, and it is used for rendering

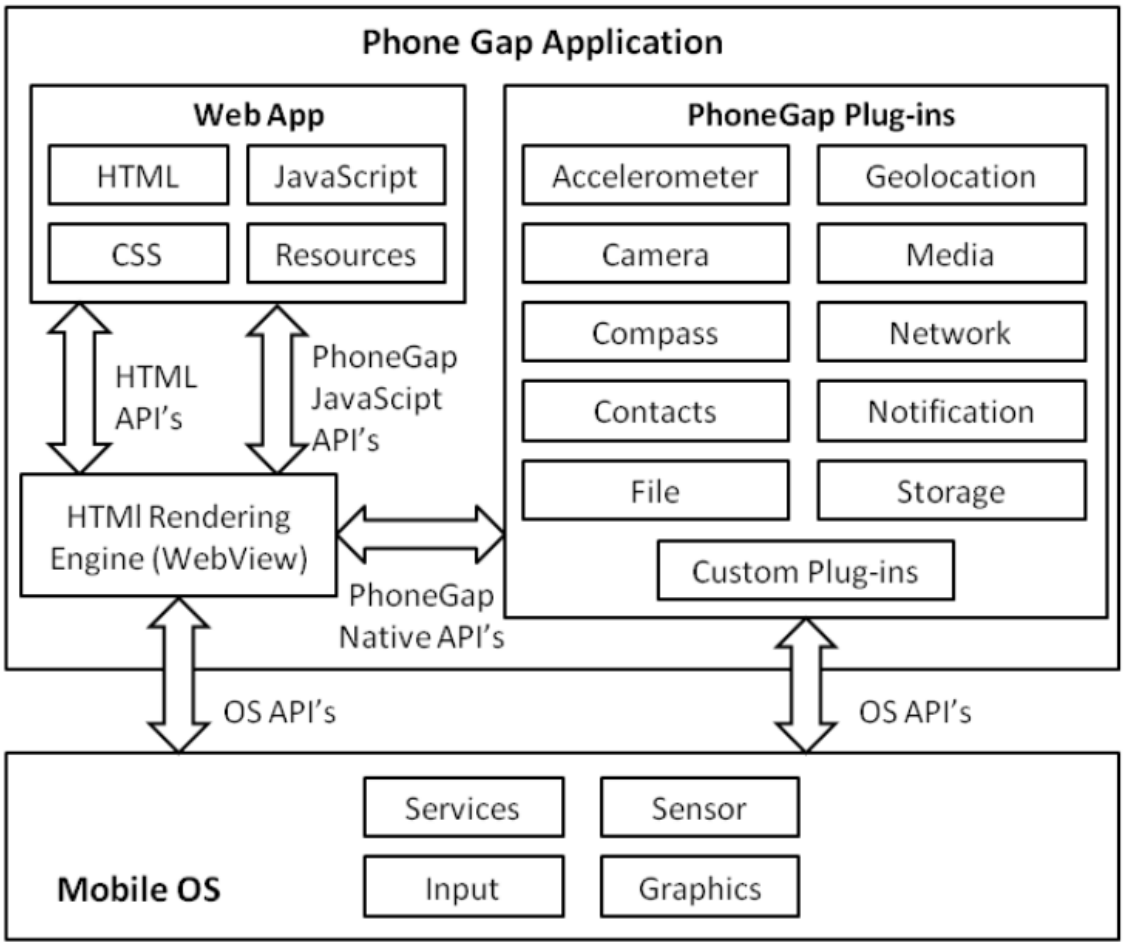

Figure 2.4 *PhoneGap architecture [38]*

HTML and CSS. The browser component allows the developer to register different events to customize the applications behavior. [41]

2.4.2 Native functionality

Web browsers do not have access to native functionality in mobile devices. This prevents a web application from using many of the benefits a mobile device can offer. The solution provided by PhoneGap is to have a native feature implemented by a plugin. The operating system interacts with a PhoneGap application either through the WebView or by a feature implemented by a plugin as seen in Figure 2.4

Plugins are packages of code, that make it possible for the WebView component to

communicate with the operating system it is running on. This is how the WebView is given access to features that a browser does not have. PhoneGap provides plugins such as storage, notifications, contacts, and accelerometer. Other plugins can be found, online and there are different registers for PhoneGap plugins. [12]

Plugins implement a native feature for at least one platform. A plugin has a single JavaScript interface that the application uses. This method hides the native code implementation so the application developers do not need to understand each platforms implementation. Developers can create their own plugins if there is no implementation for the feature they need. This can be necessary when an application need to perform actions while it is running in the background. [12]

3. TEST AUTOMATION FOR MOBILE APPLICATIONS

Testing mobile applications can be challenging due to the special requirements set by mobile devices. They should be able to operate anywhere and at any time. Depending on the number of targeted devices the application needs to function correctly on a different combination of display sizes, battery life, operating systems, computing power, etc. [24]

In this chapter we will first introduce different testing environments and their benefits and challenges in section 3.1. Then we will explain testing frameworks and what kind of testing frameworks are used for testing mobile applications in section 3.2. Finally the different types of frameworks are described in section 3.3.

3.1 Testing environments

Gao et al. [24] identified four popular infrastructures used for testing mobile applications. These are emulation, cloud, device, and crowd based testing. The different approaches can be seen in Figure 3.1. These infrastructures will be introduced in the following subsections.

Testing on physical devices and emulators have been the traditional methods for testing mobile applications. Cloud and crowd based testing have grown in popularity as the number of different mobile devices have made it unrealistic to acquire or emulate all of them.

3.1.1 Physical devices

Testing on a real mobile device is the most reliable way to test device-based functions and device specific behavior which other approaches are not able to test. This

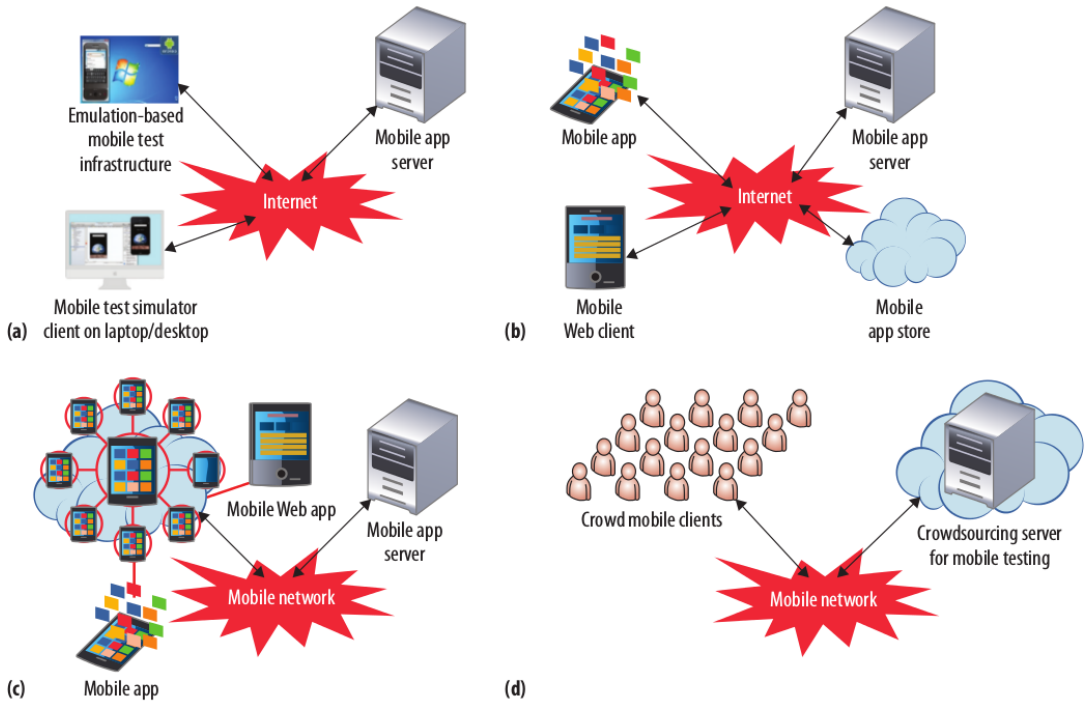

Figure 3.1 *Mobile test infrastructures: (a) emulation, (b) device, (c) cloud, and (d) crowd [24]*

approach is also costly since it requires acquiring a large number of real devices. Using real devices also means that new devices need to be added as they are made available in order to make sure the application is able to function as it should on the new device. [24]

Because of the large number of device vendors and different devices, it is unrealistic for developers to have all of the different devices. Still, testing applications on real devices with different screen sizes is recommended before the application is made available for the end users. [9]

3.1.2 Emulators

Emulators are virtual devices that are used to simulate mobile devices. They can be used to prototype, develop and test applications on a device without the actual physical device. Emulators can be used to mimic features of a real device with some limitations. For example an emulator might be able to mimic the accelerometer but not phone calls of an actual mobile device. [6]

This approach is much cheaper in comparison to testing with real devices. However there are limitations when using emulators. For instance testing complex gestures might not be possible with emulators. Gestures can still be mimicked when using emulators. Another challenge is the limited number of emulators available in order to simulate real devices.

Both Android and iOS have emulators, that can be used for testing applications. For Android the emulators can be created for different API levels on different screen sizes. Some other features can be configured as well such as memory, storage, and the Central Processing Unit (CPU). [6]

On iOS the emulators are referred to as simulators. There are simulators available for testing many of devices running iOS. The simulators are provided by the Xcode IDE, and it can be used to control the simulator, for example by sending mock locations, changing network speed, and changing the screens orientation. [18]

3.1.3 Cloud testing

There are many definition for cloud computing but most of them have in common the aspects that cloud computing is available on-demand, it is elastic and it uses resource pooling. Clouds can be private, public, community clouds or a hybrid cloud that combines some of the other types of clouds. Cloud computing leads to a service oriented architecture which can also be applied to testing. Software testing as a service (STaaS) gives testing support through web browsers and testing frameworks. [39]

Cloud based testing attempts to solve the problems in device based testing. While it may not be worthwhile for a company to set up its own testing environment, it can rent a testing environment in the cloud. This makes it much more cost-effective and scalable. [24]

3.1.4 Crowd testing

In crowd testing a mass of mobile device users are referred to as a crowd. A crowd-sourcing server is used for distributin the application as seen in Figure 3.1. Testing is done by outsourcing different testing tasks to the crowd. This method makes it

possible to have the application tested on many real devices with different configurations and in real life conditions. Developers can get results for different aspects such as usability, performance, localization and security. Crowd testing can make testing much more affordable and produce results in a shorter time period. [30]

The main challenges with crowd testing are tester selection, tester management, result aggregation and the incentive mechanism. When selecting testers it is important that they resemble the target audience. For a wide audience it is easier to find testers, but for some specific groups testers can be difficult to find. Ideally the tester will use the application the same way the final end user does. Tester management consist of gathering information about the testers and how they use the tested application. In test result aggregation all the data from the testers is gathered and analyzed. Analyzing results from a large crowd can be challenging since the results consist mostly of the applications log information and exceptions from the application. An incentive mechanism is used in order to get testers to participate in testing. Most commonly money is used as an incentive but some other incentives exist as well. For example testers can get free use of paid applications. [45]

Both Google and Apple offer ways of distributing applications for groups of testers. Google uses its Play Store, where applications can be distributed to alfa and beta testers [25]. Apple does the same using an application called TestFlight, wich installs the tested application to the testers iOS device [14].

3.2 Test automation frameworks

Test frameworks often support both emulation and device based testing. In addition to testing native mobile applications, some frameworks can be used for testing mobile web applications. There are also differences in programming languages supported by testing frameworks. Some can only be used with one language while others allow the use of many different languages such as Java, Python, and JavaScript. There are many open source testing frameworks, but also a large number of frameworks that require license contracts. [24]

According to Gao et al. [24] test automation tools for mobile devices have many limitations, such as the lack of tools that are able to test applications on different platforms and different browsers. The tools usually do not follow any standards and this makes integration with other tools difficult. Another challenge is testing large-

scale concurrent mobile test operations, which are needed for testing scalability.

Many of the tools developed for testing mobile applications were developed following techniques used by tools used for testing desktop applications. Often tools for testing mobile applications require compiling an extra agent when compiling the mobile application for testing. The agent makes it possible for the testing tool to interact with the application. With these tools it is important to compile the application without the testing libraries when submitting the application into production. There are different tools for testing native, web and hybrid mobile applications and some of the tools are able to test each type [42]. [40]

3.3 Types of test automation frameworks

Test automation frameworks are defined as a set of concepts, assumptions, and practices that form a platform for automated testing. There are many types of testing frameworks, and some of the most common ones are test script modularity, test library architecture, data-driven and hybrid test automation frameworks. [36]

With test script modularity frameworks independent scripts are created in order to test an application. The scripts can represent modules, sections or functions of the tested application. The scripts are then used to create larger test in order to create a particular test case. The benefit of this approach is that it is simple and easy to maintain. [36]

The test library architecture approach is more complicated than the test script modularity framework, since the use of libraries is required. Instead of dividing the application into scripts it is divided into functions and procedures. This makes it possible to call these library files directly from the test scripts. [36]

In data driven testing, the test data consist of the input data for the application and the expected output data. The input and output data is accessed by the test scripts, but no test data is contained in the scripts. Implementing data driven test is not concidered difficult. [36]

Table driven testing is similar to data driven testing, and it is often referred to as keyword driven testing. In addition to the input and output data used for testing it also contains sets of code used by the test scripts. This approach is concidered difficult to implement but the benefit is that maintaining the test is easy. [37]

There are different ways to combine these approaches, and they are known as hybrid test automation frameworks. Hybrid test automation frameworks attempt to use the best features of all or some of the testing framework approaches. This allows using other frameworks for tasks that can be difficult when using just one of the frameworks. [37]

4. EVALUATED TEST AUTOMATION FRAMEWORKS

In this chapter we will first list the requirements for the frameworks to be evaluated in section 4.1. Then we will introduce the selected frameworks to be evaluated in section 4.2. Finally a summary of the chapter is presented in section 4.3.

4.1 Framework requirements

There are many frameworks available for testing mobile applications, and the following requirements are used to select the frameworks that will be evaluated. The frameworks that will be compared should be open source. There should also be support available, either by documentation, or by a community.

There are frameworks for testing Android, iOS, or both. The framework needs to support testing Android applications, since Android is the most popular operating system available. With Android it is possible to test many more physical dimensions, which is an important aspect when testing hybrid applications. This is because the layout of the user interface can change with different screen dimentions, and it is important that the content is placed correctly. If a framework is able to test both iOS and Android application it is preferable that the same code can be used with little or no changes.

Some frameworks require the use of a specific IDE. The framework should be such that it does not require the use of an IDE. It should be possible to use any text editor to write test cases for the evaluated framework.

It should also be possible to run the test on emulators and real devices. Some frameworks can run test on multiple devices at the same time, but this is not a requirement.

4.2 Chosen frameworks

A list of frameworks for mobile application testing was provided by Gao et al. [24]. More information about those frameworks was searched online. Three frameworks that met the listed requirements were selected to be evaluated. Frameworks that offered the possibbility of testing iOS applications in addition to Android applications where chosen, since the possibility of testing both platforms is beneficial.

The selected frameworks for evaluation are Calabash, Appium and Selendroid. All of them are open source and they can be used to run test on emulators and real devices. Selendroid can be used to test only Android applications while Calabash and Appium can also test iOS applications. The frameworks to be evaluated are introduced in the following subsections. Their architectures are described and some code examples are also provided.

4.2.1 Calabash

Calabash is an automated UI acceptance testing framework developed by Xamarin. It can be used to test Android and iOS native and hybrid applications. By default Calabash uses the Cucumber framework which has test that are written using Gherkin, but it is also possible to use any Ruby based testing framework. Cucumber is a generic framework that manages running the tests. It uses an automation library which allows it to execute test on a specific platform. This makes it possible to write test for different platforms using Cucumber. [43]

Cucumber's Domain Specific Language (DSL) is called Gherkin. Gherkin was designed to be easily understandable by project team members with different technical bakgrounds. This is why Gherkin is a near-natural language, which has a syntax that does not require any technical understanding from its user. Gherkin consist of grammar rules that allow using natural language to specify the behavior being tested. [33]

The Calabash technology stack can be seen in Figure 4.1. The feature definitions that contain the scenarios are writting using Gherking. The step definitions that describe each step are written using Ruby. The test can be written for any platform that has an automation library providing support for the tested platform. On Android the test are run by a test server built by Calabash. This test server needs to be signed with the same certificate as the tested application. [43]

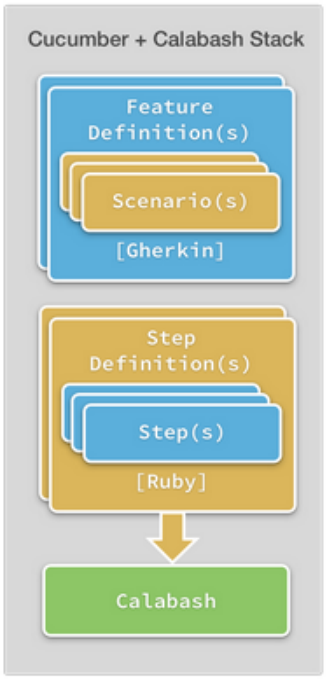

Figure 4.1 *The Cucumber technology stack [43].*

An example of a tested feature using Gherking can be seen in Program 4.1. The keywords have been highlighted. A tested feature can have one or more tested scenarios in it. In this example there is one scenario that is described on line 4. The feature has an optional description which can be seen in line 2. The steps in the test on lines 5-7 are either predefined steps or they have been implemented by the tester. The steps should be descriptive about what the effect of the step is. [43]

```
1 Feature: Credit card validation.
  Credit card numbers must be exactly 16 characters.
3
  Scenario: Credit card number is too short
5   Given I enter 7 numbers for the credit card
    And I touch the "Validate" button
7   Then I see the text "Credit card number is too short."
```

Program 4.1 *Example of a feature written in Gherkin [43].*

Calabash has many predefined step definitions that can be used for common task such as entering text, finding elements, scrolling on the screen, etc. For other task the step definitions are written using Ruby. An example of one step defined using

Ruby can be seen in Program 4.2. This step is called by line 5 in Program 4.1. On line 1 the method is defined so that it matches the Gherkin step. On lines 3 and 6 elements are found and touched. On line 5 a method for using the devices native keyboard is used. The benefit of writing custom step definitions is that they can execute faster. This is because predefined steps usually use wait times, whereas writing custom steps allows for optimization and delays can be avoided. [43]

```
1 Given(/^I enter (\d+) numbers for the credit card$/)
  do |number_of_digits|
3   touch("SystemWebView css:'#credit-card'")
    wait_for_keyboard
5   keyboard_enter_text("9" * number_of_digits.to_i)
    touch("SystemWebView css:'#validate-btn'")
7 end
```

Program 4.2 *A step definition written with Ruby [43].*

Calabash has been developed to be used with behavior driven development (BDD), but it does not require it to be used. With BDD the applications code is written only after the applications externalities have been defined. The approach is based on test-driven development, where the test describe the developed API. But instead of describing the API, the tests describe the behavior of the application. With BDD the intent is to develop software from the product owners perspective. [43]

4.2.2 Appium

Appium is a test automation framework that makes it possible to test native, hybrid and mobile web applications. Both Android and iOS applications are supported, and there is no need to modify test scripts for different platforms. [2]

The tested application does not need to be modified or recompiled, and therefore the SDK is not needed for writing test. Some frameworks require recompiling the application and then the tested application is not the same one as the application that is used in production. With Appium test scripts can be written in many different programming languages which include Ruby, Python, Java, JavaScript, PHP and C#. [2]

Appium uses a client-server architecture. The server offers a representational state transfer (REST) API for the client to use. The messages sent by the client to

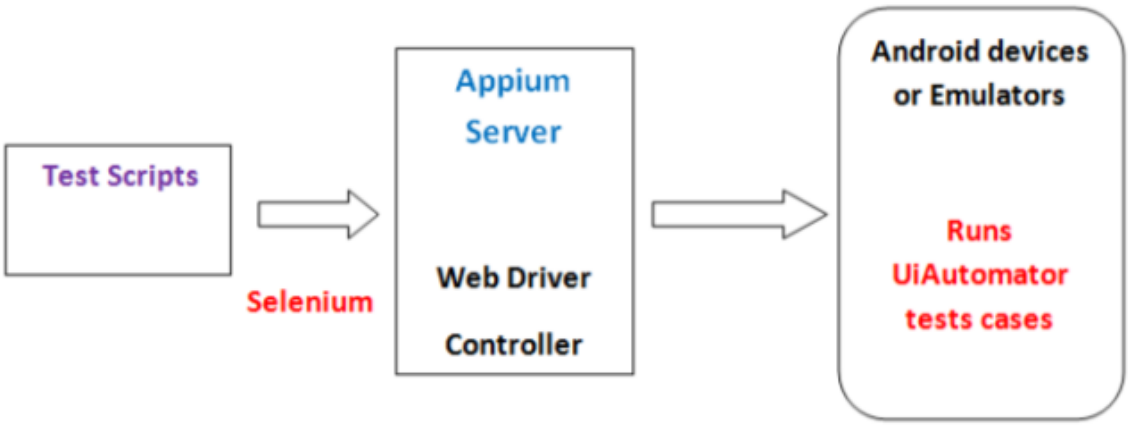

Figure 4.2 *Appium architecture [42].*

Appium contain commands. The commands are then executed by Appium on the mobile device or emulator. The server also responds with Hypertext Transfer Protocol (HTTP) responses which tell the client if the given commands were executed successfully or not. The client being used can be written in any programming language and this is why there are many programming languages availabe for Appium. Because Appium works like a web server, it can also be running on a separate machine than where the tests are running. This makes it easy to use cloud based testing. [2]

The Appium architecture for testing Android applications can be seen in Figure 4.2. The appium server runs using Node.js, which is a JavaScript runtime environment. The test scripts are written using Selenium web driver libraries and APIs. With Android the UiAutomater is used to run the test cases in the emulator or real device. [42]

An example of an Appium test written using JavaScript can be seen in Program 4.3. In line 1 the test is given a description of what is being tested. Then the drivers context is set for testing the WebView in line 5. In line 7 an element is found using its id and then the click event is trigger on the next line. In line 10 the source method is used to get the content of the view. In line 11 the include method is used to check if the given string is found in the source.

```
1 it("should navigate to new group view", function () {
    return driver
3     .contexts()
      .then(function (ctxs) {
5       return driver.context(ctxs[ctxs.length - 1]);
      })
7     .elementById('add-group-btn')
      .click()
9     .sleep(1000)
      .source().then(function (source) {
11      source.should.include('New group');
      });
13 });
```

Program 4.3 *Example of an Appium test using JavaScript.*

4.2.3 Selendroid

Selendroid is a framework used for testing Android native and hybrid applications. The framework is based on the Android instrumentation framework. With Selendroid the application being tested does not need to be modified, so the tested application can be the same one that is used in production. It is also possible for Selendroid to communicate with multiple devices and emulators at the same time. [1]

Selendroids' architecture can be seen in Figure 4.3. The WebDriver Client is also known as the Selendroid client, which is a Java client library that communicates with the Selendroid Standalone component. The Selendroid Standalone component contains a HTTP server and the Selendroid standalone driver. The Selendroid standalone driver handles communication between the selendroid-client component and the selendroid-server. It is also responsible for installing the application to be tested and the selendroid server on the device or emulator used. The selendroid server is installed alongside the tested application and it handles common activities such as taking screenshots. [1]

Selendroid tests can be written in programming languages that have a Selenium client binding available such as Java, Ruby and Python. An example of a Selendroid test written using Python can be seen in Program 4.4. When testing hybrid applications the WebView needs to be set as the context as in line 2. Examples of

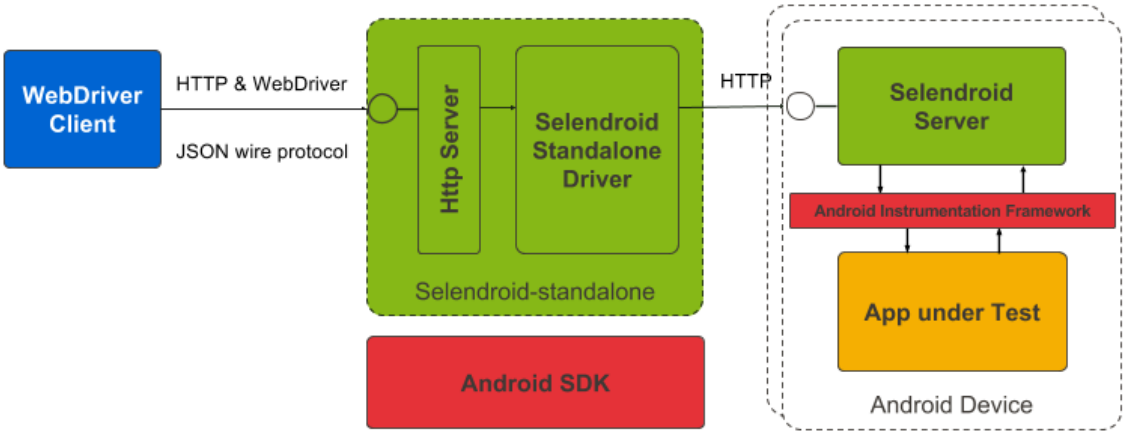

Figure 4.3 *Selendroid architecture [1].*

finding elements using an elements id or class can be seen in lines 5, 8 and 11.

```
1 def test_add_new_group(self):
      self.driver.switch_to_window('WEBVIEW')
3     self.driver.implicitly_wait(5)

5     title = self.driver.find_elements_by_class_name('navbar-inner')
      self.assertEqual('Awesome App', title[0].text)
7
      btn = self.driver.find_element_by_id('add-group-btn')
9     btn.click()

11    input_field = self.driver.find_element_by_id('group-name')
      input_field.send_keys('test input')
```

Program 4.4 *Example of a Selendroid test written using Python.*

4.3 Summary

All of the frameworks are able to test Android applications, and Calabash and Appium are able to test iOS applications as well. This is a valuable feature, if the test scripts do not need to be modified when testing on the other platform. All of the frameworks are able to run tests on real devices and emulators, which is important when attempting to cover a wide range of targeted devices without acquiring real devices.

None of the frameworks require the use of a specific IDE for writing test caces, but some of the frameworks offer additional tools, such as test case recorders to aid in writing test. A test case recorder is able to record the actions done by the tester and then generate code that will repeat those actions. With Calabash Ruby is used to write test, while Selendroid and Appium offer many programming languages for the tester to choose from. This makes it easier to adapt a framework that uses a programming language already known by the tester.

5. EVALUATION CRITERIA

In this chapter, the criteria for evaluating the frameworks are introduced. In section 5.1 the aspects for evaluating test implementations are introduced. The methods used for evaluating the user interface are described in section 5.2. In section 5.3 the native features to be evaluated are examined.

In section 5.4 the ways of evaluating how a framework is able to test an applications lifecycle are described. The methods for evaluating platform support are presented in section 5.5. Finally in section 5.6 the methods for evaluating the frameworks documentation and community support are described.

5.1 Test implementation

One aspect when evaluating test implementation is on the basis of how much work is required for creating a test. Frameworks often have predefined methods for testing common functionality, such as clicking a button or scrolling the screen. Features that do not have predefined functions require the tester to implement them. The frameworks are evaluated on how easy it is to implement test for more complex features.

Another aspect is how widespread the technologies used by the framework are. If a framework offers multiple programming languages for writing test, it may be easier for developers to use a framework that does not require learning a new programming language. It is also easier to find solutions for problems that arise when using widespread technologies.

Some frameworks offer additional tools that help with creating tests. A common tool is a test recorder, which records the input given by the user. Then the recorded input is formed into the testing frameworks test script. If a framework offers additional tools they can be used to help implementing tests.

5.2 Testing the user interface

The user interface is used to interact with the user and to display content to the user. The frameworks are evaluated in terms of how well they are able to test the application on different screen sizes, and if they are able to validate that the content is correct. They are also evaluated if they are able to use gestures that are commonly used with mobile applications.

5.2.1 Different screen sizes

Testing that the application works on different screen sizes, is done by testing the application on multiple emulators with different screen dimensions. Applications may have been made to display the content in a different way depending on if the application is running on a phone or a tablet, and depending on the screens orientation. Testing the application on different screen orientations and checking if it behaves correctly when the orientation changes is important, because users may change the orientation of the device in any view in the application.

With some applications or some views in an application, the screens orientation may be locked in order to prevent the screen from changing from landscape to portrait, or vice versa. The framework should be able to test which screen orientation is active, and if it remains the same when the rotation has been locked by the application.

An application that is running on a device that has a large screen is often designed to display content differently, than it would on a device with a small screen. In this case the content may reflow in to the available space. An example of UI reflow can be seen in Figure 5.1. In the narrower layout all the content is stacked, while on a wider UI the content is placed differently. The frameworks are evaluated in terms of if they are able to validate if elements are positioned as they should be in different screen sizes, and if they change their positions correctly in different screen orientations.

Instead of rearranging the flow of the content on different screen sizes and orientations, it is also possible for an application to transform the content differently. In Figure 5.2 an example of transforming content on different screen sizes is shown. On the left a mobile device uses a menu for displaying a navigation menu. On the right side is a wider screen that uses tabs to display the same navigation items. The

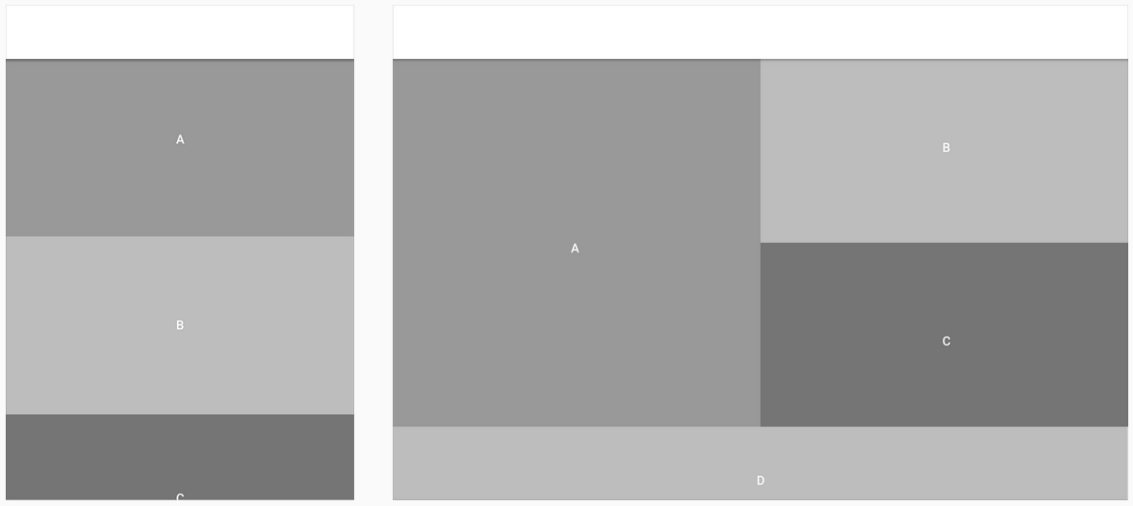

Figure 5.1 *An example of UI reflow [28].*

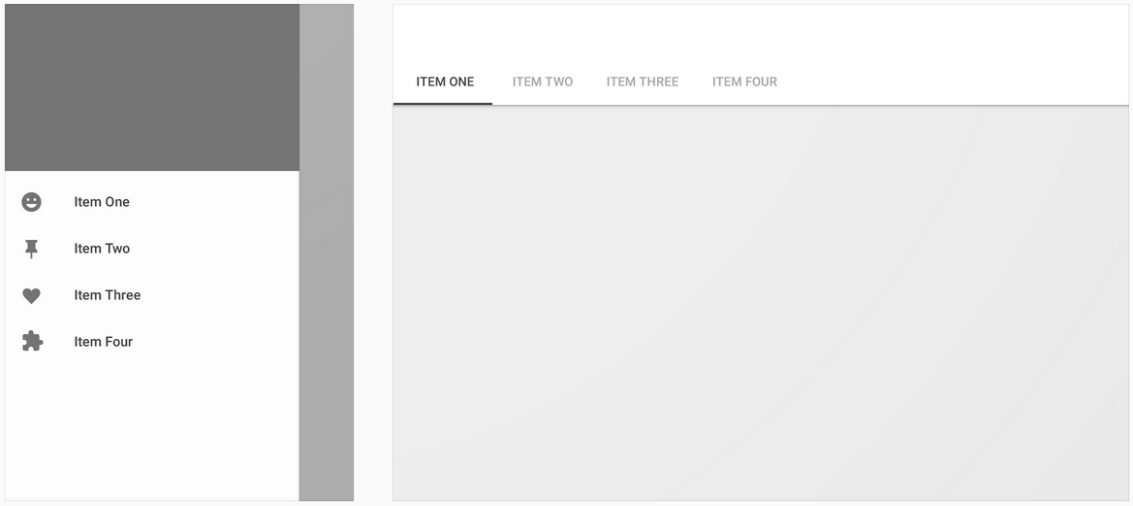

Figure 5.2 *An example of sidenavigation transforming into tabs [28].*

frameworks should be able to check that the correct content is shown in different screen orientations.

5.2.2 Validating content

The content displayed to the user is commonly text and images. Errors and messages may be displayed to the user using different colors and icons. These messages can be visible on the screen for a limited time and a message can have a short delay before

becoming visible on the screen. It should be possible to check that the content in these types of messages is correct. The content that is being validated might be available in the active screen, but it might not be visible on the screen. It should be possible to scroll on the screen until the element is visible to the user.

The frameworks are evaluated in terms of if they are able to validate if the text displayed by the application is correct. Images are either in the devices memory or they are downloaded from the Internet. The frameworks are evaluated if they are able to validate if an image is displayed after it has been loaded.

5.2.3 Gestures

Users can use different gestures to interact with an application. Being able to test common gestures is a requirement for the frameworks. Common gestures are touching elements, dragging elements, and swiping the screen. Some of these gestures are evaluated in terms of if a frameworks is able to use a gesture in tests. More complex gestures are evaluated separately, and they are evaluated in terms of how difficult it is to create custom complex gestures.

Swiping the screen is commonly used to reveal content that is not visible. One example is showing a navigation menu on one side of the screen when a swipe starts from the side of the screen towards the center of the screen. The frameworks are evaluated in terms of if they are able to open such a navigation menu by swiping the screen.

A common way of refreshing content in an application is by pulling down on the view and letting go. This is referred to as the pull to refresh gesture. An example of the gestures behavior can be seen in Figure 5.3. In the first screen the list is pressed and in the second screen the list is dragged down. The third screen shows the loading indicator after the user has let go of the screen. The circle in the first two images show the position where the user is pressing the screen. This is a common feature, which the frameworks are evaluated in terms of if they are able to test it.

It is beneficial if a framework is able to test more complex gestures, where, for example, more than one finger is needed. The frameworks are evaluated in terms of how they are able to execute a pinch gesture. When zooming in or out of images, a pinch gesture is commonly used instead of zooming in and out using buttons. In Figure 5.4 the movement of the users fingers are shown when pinching the screen.

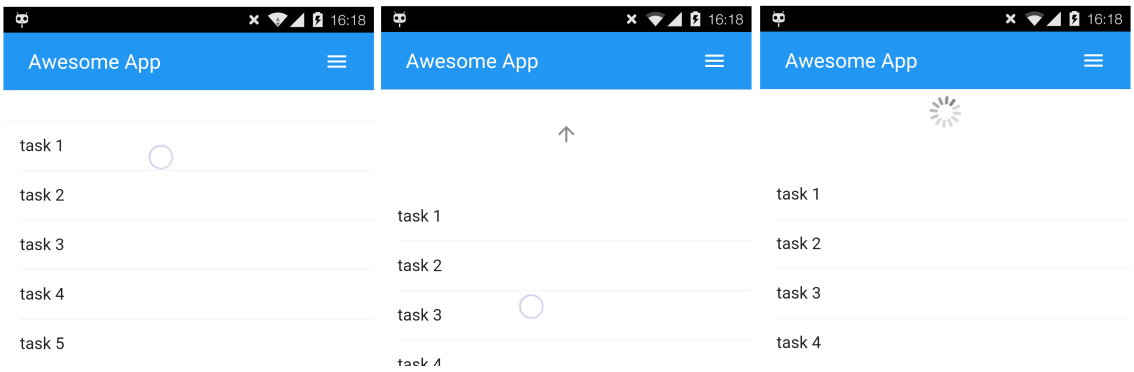

Figure 5.3 *Pull to refresh gesture.*

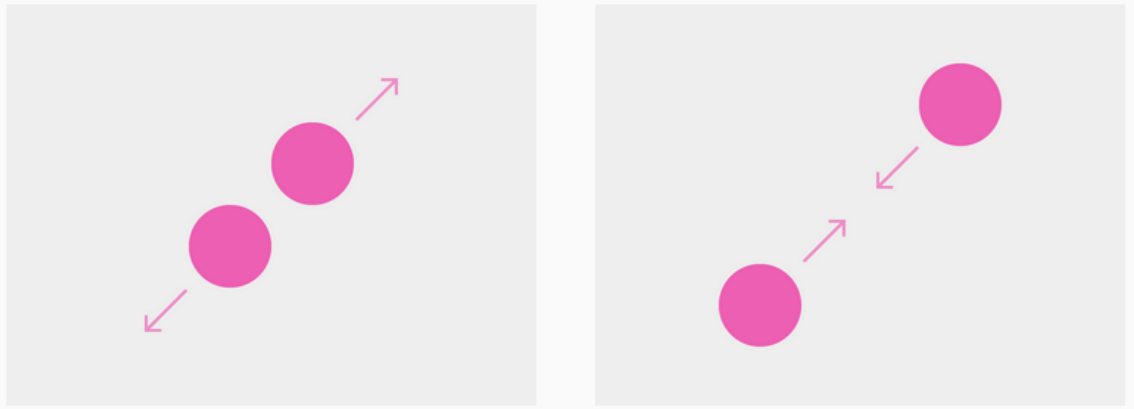

Figure 5.4 *The pinch gesture [28].*

5.3 Testing native features

The native features to be evaluated are testing the devices camera, mocking the location of the device, and validating and interacting with notifications. These features were selected because they are commonly used features in mobile applications. Some of them may behave differently in different operating systems, and the test may need to have alternative behaviors defined for testing other platforms. It is also possible that a different test is written to test the feature on another operating system.

5.3.1 Camera

Taking photos is a commonly used feature with mobile devices. Therefore, being able to write test where the camera is used is necessary. A hybrid application created

with PhoneGap uses a camera plugin for accessing the camera.

When activating the camera, the application opens the devices native camera view. The camera view is no longer a WebView and the elements are not located using the same methods as in the WebView. It is necessary for the framework to be able to take a photo in this view. The framework may have a predefined method for using the camera, or it should at least be possible to implement a method for testing this feature.

5.3.2 Location

Mobile devices can provide to the application their location, which is based on GPS or the network. This might be used by an application to show the users location on a map, measure distances between positions, or perform specific task when the device is in a certain location. There can also be a different set of behavior for an application when the location is not provided as the operating system can not access the location. In some cases the accuracy of the provided location is important and only accurate location information is usable.

With Android and iOS it is possible to mock the location of the device or emulator. The frameworks should be able to mock the location by allowing the tester to provide longitude and latitude coordinates. The movement of a user is simulated by giving a new set of coordinates after a given number of seconds.

5.3.3 Notifications

Notifications are used to inform the user by showing a message, playing a sound, or by showing an icon in the devices status bar. The notification can be shown whether the application is active or not. In many cases, notifications are only shown when an application is not active.

The frameworks are evaluated in terms of if they are able to check if a notification has appeared, and if the framework is able to validate that the content in the notification is correct. Pressing the notification when the application is in the background makes the application active. An application can have different behavior when activated by a notification and it is beneficial if a framework is able to press a notification when the application is in the background.

5.4 Testing the application lifecycle

Applications may have task they need to execute at different phases of the applications lifecycle. An application returning from the background might need to check a server for information, or an application being terminated might need to save information before shutting down. There are differences with the lifecycle of an Android application compared to an iOS application. But with PhoneGap applications the application can have functionality when the application starts, resumes, or is paused.

The frameworks are evaluated if they are able to suspend and close the application. They are also evaluated based on if they are able to start the application after closing or suspending the application. Both Android and iOS have a home button that should be used for setting the application to the background. This makes it possible to suspend the application in any view the user is in, and it is preferable if this can be used to set the application to the background.

5.5 Platform support

All of the frameworks support testing Android applications. Calabash and Appium support testing iOS applications and they are evaluated in terms of how easy it is to set up the framework for testing iOS applications, and how easy it is to reuse the code on another platform.

Reusing the test code for testing another platform is evaluated on how little if any changes are needed for the test scripts to work. In addition, there can be cases when an application needs to behave differently on one platform. In this case the framework is evaluated in terms of how easy it is to either use a different test on certain platforms or how easy it is to have a test behave differently when testing a certain platform.

5.6 Documentation and community support

In order to use the frameworks, the documentation should cover what is needed in order to set up the environment for testing applications. In addition the documentation should explain the basic features of the framework and contain examples of

how to write tests. The frameworks are open source and it is important that the documentation is up to date as the framework develops.

Community support is important for finding solutions for less common tasks that are not covered by the documentation. An active community is more likely to provide solutions for these tasks. Community support will be evaluated in terms of if solutions can be found for problems during test implementation and if the community has active discussion groups.

5.7 Summary

The criteria to be evaluated are listed in Table 5.1. The frameworks are given points between zero and three for each criterion. Zero points are given for example, when a framework is not able to implement a test for a feature. One point is given when it is possible, but with difficulties. If the feature is possible with little effort it is given two points. Three points are given when the criterion has the best possible solution, for example, if the framework provides predefined methods for complex gestures.

Table 5.1 *Evaluated criteria*

Criterion	Description
Test implementation	How difficult is it to write tests?
Screen sizes	Ability to use multiple emulators. Checking the contents position.
Validating content	How easy is it to find and assert content?
Simple gestures	Pressing buttons and swiping the screen.
Complex gestures	Implementing the pinch gesture.
Camera	Is the framework able to use the camera?
Location	Is the framework able to send mock locations to the emulator?
Notifications	Is the framework able to check the notifications?
Application lifecycle	Is the framework able to close, suspend and resume the application?
Code reuse	How difficult is it to use the same code on another platform?
Documentation and community support	Is there documentation available? How active is the community?

6. EVALUATION RESULTS

In this chapter we will go trough the results for the criteria defined in chapter 5. In sections 6.1- 6.11 the results for each of the criterion are presented. In section 6.12 a summary of the results is presented, and the final score of each evaluated framework is shown. Finally in section 6.13, the evaluation of the frameworks is evaluated on how well it succeeded and what could have been done differently.

6.1 Test implementation

Calabash has a console that allows the tester to interact with the application using the methods provided by the API. This made implementing tests faster, as running an entire test was not required in order to see the how the method interacted with the application. Calabash made the reuse of common steps easy, and it was possible to define common steps for all of the scenarios within a feature. Calabash is given three points for test implementation.

Appium has an inspector tool that makes it possible to inspect the tested applications elements. The inspector can also be used to record the actions of a user, and a test that repeats the actions is generated as a result. The recorded test do not contain assertions that validate that the content is correct, and elements are usually found by using XML Path Language (XPath). The elements found using XPath are not easy to maintain, and this is why the recorded actions may need to be edited by the tester. The recorded test are useful templates for creating the tests scripts. Appium receives two points for this criterion.

Implementing tests for Selendroid required more time than when using the other frameworks. Selendroid requires Selendroid Standalone to manage installing the tested application and the test server. In many cases when a test was not able to execute a command the test did not stop correctly. It was then necessary to restart Selendroid Standalone and it was not always able to connect with the device being

used for testing. This made implementing the test slow. Selendroid is given one point for test implementation.

6.2 Screen sizes

All of the frameworks were able to switch the screens orientation and validate if the elements where in the correct positions. None of the frameworks had predefined methods for validating the positions of elements, but all of the frameworks were able to find the coordinates of elements, and implementing comparisons was not difficult.

Acquiring the screens dimensions was not difficult for any of the frameworks. The dimensions can be used to confirm where the elements should be position for the given dimension. With Calabash it is possible to only run specific features or scenarios based on a profile created for testing. This makes it possible to tag features and scenarios that will be tested only on certain devices. Different tests for phones and tablets could for instance be tagged in order to test the behavior of the application in a different way when running on a phone or a tablet. Calabash is given two points for this criterion.

Both Appium and Selendroid have methods for rotating the screen, and checking the current devices dimensions did not present difficulties. Both Appium and Selendroid receive two points for this criterion.

6.3 Validating content

Calabash uses a query method that can be used to find elements using CSS selectors. There were many predefined methods for validating content. Some of the methods can be used to wait until the expected content becomes visible, or until it is no longer visible. These methods prevent the tester from having to use methods that make the test wait for a number of seconds and then check for the content. With Calabash, in addition to logging an error message when a test fails, it was also possible to take a screenshot when the test failed. This feature can make it easier to find the problem with the application, as the tester can check what was visible when the test failed. Calabash receives three points for this criterion.

Appium has similar methods that found elements using CSS selectors and methods that checked that the content was correct. Appium receives two points for this

criterion. Selendroid was also able to use CSS selectors to find elements, but there were no methods for waiting for content to become visible. Selendroid receives one point for this criterion.

6.4 Simple gestures

Performing simple gestures was possible with all of the frameworks. Calabash has predefined steps that are able to perform basic gestures, such as swipe, double tap, and long press. With Appium gestures can be created by the tester by combining simple touch actions. Selendroid has similar method as Calabash for double taps, and long presses, etc.

Testing applications would not be possible if it would not be possible to use simple gestures. This is why it is critical that all of the framworks have the possibility of using simple gestures. None of the frameworks had difficulties with this criterion. All of the frameworks are given three points for this criterion.

6.5 Complex gestures

Calabash has predefined methods for many complex gestures such as the pinch gesture. Many of the gestures can be given parameters, for example the swipe methods start and end points. Calabash is given two points for this criterion, because it is possible to use complex gestures, but it is not possible for a tester to create their own touch actions.

Appium makes it possible to define complex gestures by defining many individual actions, that are then performed at the same time. This allows the tester to create actions that use more than one finger. This makes it possible to test any complex gesture, and therefore Appium can even be used to test drawing on the screen. Appium is given three points for this criterion.

Selendroid was not able to use complex gestures using Python. Multitouch support is only available for the Java client. Because of this Selendroid should be used with Java if complex gestures is a requirement. Selendroid is given one point for this criterion, because it is possible to use multitouch, but not with all of the available programming languages.

6.6 Camera

None of the frameworks had a predefined method for using the camera. It was possible to create methods for using the camera with Calabash and Appium. With Android the Android Debug Bridge (ADB) was used to operate the cameras buttons. This required keycodes for taking a photo, and another keycode for accepting the photo. System commands can be useful for testing other features as well. They can be used for pressing volume buttons, the return button, the home button, etc. After taking a photo in Android, the user is presented with a view where they can accept the photo. In this view the button for accepting the photo is different in Android devices and it may require the test to try the different possible buttons in order for it to work correctly.

Calabash and Appium receive two points for this criterion. Selendroid receives zero points, because it was not possible to use the camera. Since Selendroid is not able to send keycodes using ADB, it may be limited in testing other inputs the user may give, such as increasing the volume.

6.7 Location

With Android, mocking the location required adding a permission to the tested application in order to mock the devices location. Calabash and Appium were able to mock the location of the device. Calabash uses a geocoding library, that allows the tester to use location names in addition to GPS coordinates. These make the test more readable compared to using coordinates. Calabash receives three points for this criterion.

Appium was able to mock the location using coordinates and it receives two points for this criterion. Selendroid was not able to mock locations and it is given zero points for this criterion.

6.8 Notifications

None of the frameworks have methods that make it possible to check the content of the notifications. With Android it is possible to check the contents of the notification drawer using ADB. This makes it possible to validate that the applications

notification has been shown and that it has the correct message, but using it to open the application and confirming that the application behaves as it should is not possible.

Appium was the only framework that was able to open the notification drawer. It was not able to find the location of the notification and this prevented selecting the notification. Appium and Calabash receive one point for this criterion, because they are able to validate that the notification has been shown and that the content is correct. Since Selendroid is not able to use the ADB it was not able to validate the notifications and receives no points for this criterion.

6.9 Application lifecycle

Calabash has a predefined method for moving the app in to the background, but when tested the framework did not have the method implemented for testing Android applications. It was possible to move the application to the background using the ADB with Android. It was not possible to resume the application when it was running in the background, but it was possible to restart the application. Calabash is given one point for this criterion.

Appium had the same limitations as Calabash and was only able to set the application to the background before restarting it. Appium also receives one point for this criterion. With Selendroid it was possible to get the application to the background, but returning to the application was not possible. This feature could be possible to implement using another programming language. Selendroid is given zero points for this criterion.

6.10 Code reuse

Calabash and Appium are both able to run tests on iOS, but both of them have many limitations when trying to reuse code written for testing Android. With Calabash the API used for writing tests for Android is different from the API for testing applications in iOS. Using a test written for Android might require creating an alternative behavior for the other platform. Many of the methods are the same for both APIs and the tags used by Calabash make it possible to define test scenarios that are skipped using different profiles. In this case profiles can be defined for

Android and iOS, which allows skipping some scenarios that are not possible to test, or the tests are not made for some other reason. Calabash receives two points for this criterion.

Appium was able to use many of the tests written for testing Android. There were some limitations as all of the methods used by the scripts were not implemented for both operating systems. Appium does not have a similar way of using test profiles as Calabash, and therefore it requires more effort to divide the test based on which platforms they will be used on. Appium is given one point for this criterion. Selendroid is not able to test iOS applications and is given zero points.

6.11 Documentation and community support

Calabash has documentation that is maintained as part of the project, but there is also documentation that is maintained by Xamarin. Xamarin maintains and develops Calabash. Since Calabash uses the Cucumber framework, the documentation for Cucumber is also important, and following different sets of documentation can be difficult. Some of Calabashes documentation was not up to date, as some methods had been either removed or renamed in newer versions. The community is active and many issues were solved by searching for the issue in the community's discussion forum. Calabash receives two points for this criterion.

Appium is documented well, and it also has examples for all of the programming languages that can be used with it. The documentation did not mention if a feature had not been implemented, and it was only possible to notice when running a test that used the feature. This could be because a feature may be implemented on one platform but not in the other. The community is active and the discussion forum for Appium was useful for solving common problems. Appium receives two points for this criterion.

Selendroid has very little documentation, especially for testing cross-platform applications. Examples were also limited, and some programming languages had more examples than others. Community support was also limited. Selendroid is given one point for this criterion.

6.12 Summary

The points given for each criterion for the frameworks can be seen in Table 6.1. Calabash received the highest score between the evaluated frameworks and Appium did not fall far behind. With Selendroid there were challenges in testing native features, and this resulted in Selendroid receiving a much lower score than the other frameworks.

Based on these results Calabash is recommended for automating test for cross-platform mobile applications. Appium was able to test complex gestures better than Calabash. While Calabash has a large number of predefined methods for testing complex gestures it does not give the tester the ability to create custom complex gestures as easily as Appium. Out of the evaluated frameworks, Calabash offered the best solution for reusing the test code on another platform.

Table 6.1 *Evaluation results*

Criterion	Calabash	Appium	Selendroid
Test implementation	3	2	1
Screen sizes	2	2	2
Validating content	3	2	2
Simple gestures	3	3	3
Complex gestures	2	3	1
Camera	2	2	0
Location	3	2	0
Notifications	1	1	0
Application lifecycle	1	1	0
Code reuse	2	1	0
Documentation and community support	2	2	1
Overall score	24	21	10

6.13 Evaluation of the results

The evaluation was done by writing tests for an application that was made for testing the criteria. For Android the tests were used on two real devices and two emulators. For reusing the code, the test where used with iOS on one device, and two emulators. A larger number of devices could have been used to make sure that

the test function as expected. Especially with Android, testing with many more devices that have different API levels, could have given different results.

With Appium and Selendroid all of the features have not been implemented for all of the available programming languages. This makes choosing the programming language important, and choosing the language depends on the features that need to be tested. The results could be different if the evaluation would be done using another programming language.

Overall, the evaluation provided the results needed for making a decision, when selecting a framework for testing cross-platform mobile applications. It also provided a better understanding of the limitations and challenges that are met when testing mobile applications.

7. CONCLUSIONS

In this thesis, the development of cross-platform mobile applications using PhoneGap was introduced. Three different test automation frameworks were evaluated for automating tests for hybrid mobile applications. It became clear when searching for these frameworks, that there are not many tools available for testing hybrid applications. Most of them were limited to one platform, and none were capable of testing more than the two most popular mobile operating systems available. If hybrid mobile applications become more popular, more frameworks may become available, and it would be worth evaluating their cabapilities.

While there are not many frameworks available for testing cross-platform mobile applications, there are frameworks which can be used to test a wide range of features. Testing features that are limited to operating within the application and not too dependent on the other features provided by the operating system, such as taking photos and displaying notifications, can be tested without difficulties on most of the testing frameworks. Features that behave differently in different versions of the operating systems are more difficult to test.

The frameworks can test application on both real devices and emulators. This makes it possible to test a large number of devices with different system configurations and screen sizes. Calabash is also able to use profiles that make it easier to run selected test on certain devices.

In conclusion, the intent of this thesis was to find a suitable framework for testing cross-platform applications. The framework that met the criterion with the best score was Calabash. Therefore, Calabash is recommended for automating the testing of cross-platform applications. Calabash can be used to test both Android and iOS applications. It is recommended using behavior driven development with Calabash, as it has been developed for that purpose.

BIBLIOGRAPHY

[1] *Selendroid's Architecture*, 2015, Available (accessed on 29.12.2015): http://selendroid.io/architecture.html.

[2] *About Appium*, 2016, Available (accessed on 2.1.2016): http://appium.io/slate/en/master/.

[3] S. Amatya and A. Kurti, "Cross-platform mobile development: challenges and opportunities," in *ICT Innovations 2013*. Springer, 2014, pp. 219–229.

[4] *Amazon underground*, Amazon, 2016, Available (accessed on 19.03.2016): https://www.amazon.com/gp/browse.html?node=9530541011.

[5] *Android Dashboard*, Android, 2015, Available (accessed on 13.12.2015): http://developer.android.com/about/dashboards/index.html.

[6] *Android Emulator*, Android, 2015, Available (accessed on 14.12.2015): http://developer.android.com/tools/help/emulator.html.

[7] *The Android Source Code*, Android, 2015, Available (accessed on 16.12.2015): http://source.android.com/source/index.html.

[8] *Android studio*, Android, 2015, Available (accessed on 15.12.2015): http://developer.android.com/sdk.

[9] *Using hardware devices*, Android, 2015, Available (accessed on 13.12.2015): http://developer.android.com/tools/device.html.

[10] *What is API Level?*, Android, 2015, Available (accessed on 16.12.2015): http://developer.android.com/guide/topics/manifest/uses-sdk-element.html.

[11] *Starting an Activity*, Android, 2016, Available (accessed on 16.01.2016): http://developer.android.com/training/basics/activity-lifecycle/starting.html.

[12] *Overview of Apache Cordova*, Apache Cordova, 2015, Available (accessed on 14.11.2015): http://cordova.apache.org/docs/en/5.4.0/guide/overview/index.html.

[13] F. Appiah, J. Hayfron-Acquah, J. K. Panford, and F. Twum, "A tool selection framework for cross platform mobile app development," *International Journal of Computer Applications*, vol. 123, no. 2, pp. 14–19, 2015.

[14] *About App Distribution Workflows*, Apple, 2015, Available (accessed on 19.03.2016): https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/.

[15] *App Store Review Guidelines*, Apple, 2015, Available (accessed on 13.12.2015): https://developer.apple.com/app-store/review/guidelines/.

[16] *iOS App Store*, Apple, 2015, Available (accessed on 13.12.2015): https://developer.apple.com/support/app-store/.

[17] *Xcode*, Apple, 2015, Available (accessed on 13.12.2015): https://developer.apple.com/support/xcode/.

[18] *About Simulator*, Apple, 2016, Available (accessed on 20.03.2016): https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/iOS_Simulator_Guide.

[19] *The App Life Cycle*, Apple, 2016, Available (accessed on 16.01.2016): https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/TheAppLifeCycle/TheAppLifeCycle.html.

[20] L. Corral, A. Sillitti, G. Succi, A. Garibbo, and P. Ramella, "Evolution of mobile software development from platform-specific to web-based multiplatform paradigm," in *Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, ser. Onward! 2011. New York, NY, USA: ACM, 2011, pp. 181–183. [Online]. Available: http://doi.acm.org/10.1145/2048237.2157457

[21] P. R. de Andrade, A. B. Albuquerque, O. F. Frota, R. V. Silveira, and F. A. da Silva, "Cross platform app: a comparative study," *arXiv preprint arXiv:1503.03511*, 2015.

[22] M. Diehl, "Phonegap application development," *Linux J.*, vol. 2013, no. 225, Jan. 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2436343.2436347

[23] W. S. El-Kassas, B. A. Abdullah, A. H. Yousef, and A. M. Wahba, "Taxonomy of cross-platform mobile applications development approaches," *Ain Shams Engineering Journal*, 2015.

[24] J. Gao, X. Bai, W.-T. Tsai, and T. Uehara, "Mobile application testing: a tutorial," *Computer*, no. 2, pp. 46–55, 2014.

[25] *Set up alpha/beta tests*, Google, 2015, Available (accessed on 19.03.2016): https://support.google.com/googleplay/android-developer/answer/3131213.

[26] *Google Play*, Google, 2016, Available (accessed on 19.03.2016): https://play.google.com/store.

[27] *Google Play services*, Google, 2016, Available (accessed on 19.03.2016): https://play.google.com/store/apps/details?id=com.google.android.gms.

[28] *Responsive UI*, Google, 2016, Available (accessed on 12.01.2016): https://www.google.com/design/spec/layout/responsive-ui.html.

[29] E. Grøtnes, "Standardization as open innovation: two cases from the mobile industry," *Information Technology & People*, vol. 22, no. 4, pp. 367–381, 2009. [Online]. Available: http://dx.doi.org/10.1108/09593840911002469

[30] F. Guaiani and H. Muccini, "Crowd and laboratory testing can they co-exist?: An exploratory study," in *Proceedings of the Second International Workshop on CrowdSourcing in Software Engineering*, ser. CSI-SE '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 32–37. [Online]. Available: http://dl.acm.org/citation.cfm?id=2820116.2820123

[31] D. Han, C. Zhang, X. Fan, A. Hindle, K. Wong, and E. Stroulia, "Understanding android fragmentation with topic analysis of vendor-specific bugs," in *Reverse Engineering (WCRE), 2012 19th Working Conference on*. IEEE, 2012, pp. 83–92.

[32] G. Hartmann, G. Stead, and A. DeGani, "Cross-platform mobile development," *Mobile Learning Environment, Cambridge*, 2011, Available: https://wss.apan.org/jko/mole/Shared%20Documents/Cross-Platform%20Mobile%20Development.pdf.

[33] M. Hesenius, T. Griebe, and V. Gruhn, "Towards a behavior-oriented specification and testing language for multimodal applications," in *Proceedings*

of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems, ser. EICS '14. New York, NY, USA: ACM, 2014, pp. 117–122. [Online]. Available: http://doi.acm.org/10.1145/2607023.2610278

[34] *Introduction to mobile application development*, IBM, 2015, Available (accessed on 15.11.2015): https://www-01.ibm.com/support/knowledgecenter/SSZH4A_6.2.0/com.ibm.worklight.getstart.doc/getstart/c_mobile_concepts.html.

[35] *Worldwide Smartphone Market Will See the First Single-Digit Growth Year on Record, According to IDC*, IDC, 2015, Available (accessed on 11.12.2015): http://www.idc.com/getdoc.jsp?containerId=prUS40664915.

[36] M. Kelly, *Choosing a test automation framework*, IBM, 2003, Available (accessed on 14.12.2015): http://www.ibm.com/developerworks/rational/library/591.html.

[37] C. Nagle, *Test automation frameworks*, 2015, Available (accessed on 15.12.2015): http://safsdev.sourceforge.net/DataDrivenTestAutomationFrameworks.htm.

[38] M. Palmieri, I. Singh, and A. Cicchetti, "Comparison of cross-platform mobile development tools," in *Intelligence in Next Generation Networks (ICIN), 2012 16th International Conference on*. IEEE, 2012, pp. 179–186.

[39] L. Riungu, O. Taipale, and K. Smolander, "Research issues for software testing in the cloud," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, Nov 2010, pp. 557–564.

[40] G. Shah, P. Shah, and R. Muchhala, "Software testing automation using appium," 2014.

[41] M. Shehab and A. AlJarrah, "Reducing attack surface on cordova-based hybrid mobile apps," in *Proceedings of the 2nd International Workshop on Mobile Development Lifecycle*. ACM, 2014, pp. 1–8.

[42] S. Singh, R. Gadgil, and A. Chudgor, "Automated testing of mobile applications using scripting technique: A study on appium," 2014.

[43] *Introduction to Calabash*, Xamarin, 2015, Available (accessed on 28.12.2015): https://developer.xamarin.com/guides/testcloud/calabash/introduction-to-calabash/.

[44] S. Xanthopoulos and S. Xinogalos, "A comparative analysis of cross-platform development approaches for mobile applications," in *Proceedings of the 6th Balkan Conference in Informatics*, ser. BCI '13. New York, NY, USA: ACM, 2013, pp. 213–220. [Online]. Available: http://doi.acm.org/10.1145/2490257.2490292

[45] M. Yan, H. Sun, and X. Liu, "itest: Testing software with mobile crowdsourcing," in *Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies*, ser. CrowdSoft 2014. New York, NY, USA: ACM, 2014, pp. 19–24. [Online]. Available: http://doi.acm.org/10.1145/2666539.2666569